University of Pittsburgh

Algorithms and Data Structures 2
CS 1501

Fall 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Homework 1 is due on Monday 1/18 at 11:59 pm
- Lab 1 posted on Canvas
  - Explained in recitations of this week
  - Github Classroom
- Assignment 1 will be posted tonight

Last lecture …

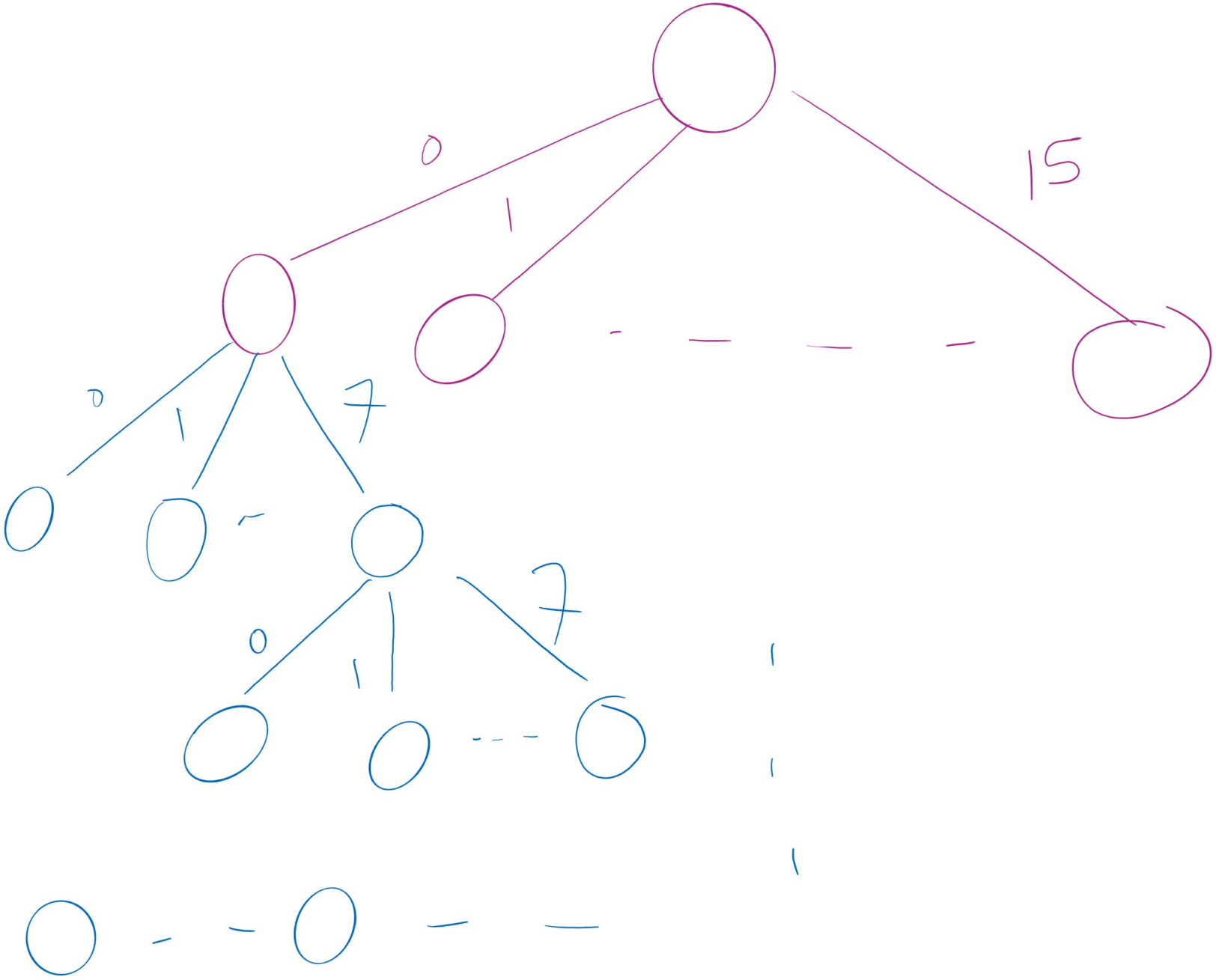
- 3-sum problem
- The $\sum_{all\ statements} Cost * frequency$ technique for modeling runtime of algorithms
- Asymptotic analysis
- Boggle Game Problem

CourseMIRROR Reflections

Boggle Game Problem (Recap)

- Words at least 3 adjacent letters long must be assembled from a 4x4 grid
- Adjacent letters are horizontally, vertically, or diagonally neighboring
- Any cube in the grid can only be used once per word

Search Space



Boggle Game: Decisions and Choices

- For the first decision (which tile to start from), how many possible choices to choose from?
- Starting from a tile, the next decision to make is which adjacent tile to move to.
  - There are at most 8 possible choices to choose from
- There is a decision to make at each tile that we go through
  - A maximum of 15 decisions

First decision

Are all 16 choices valid?

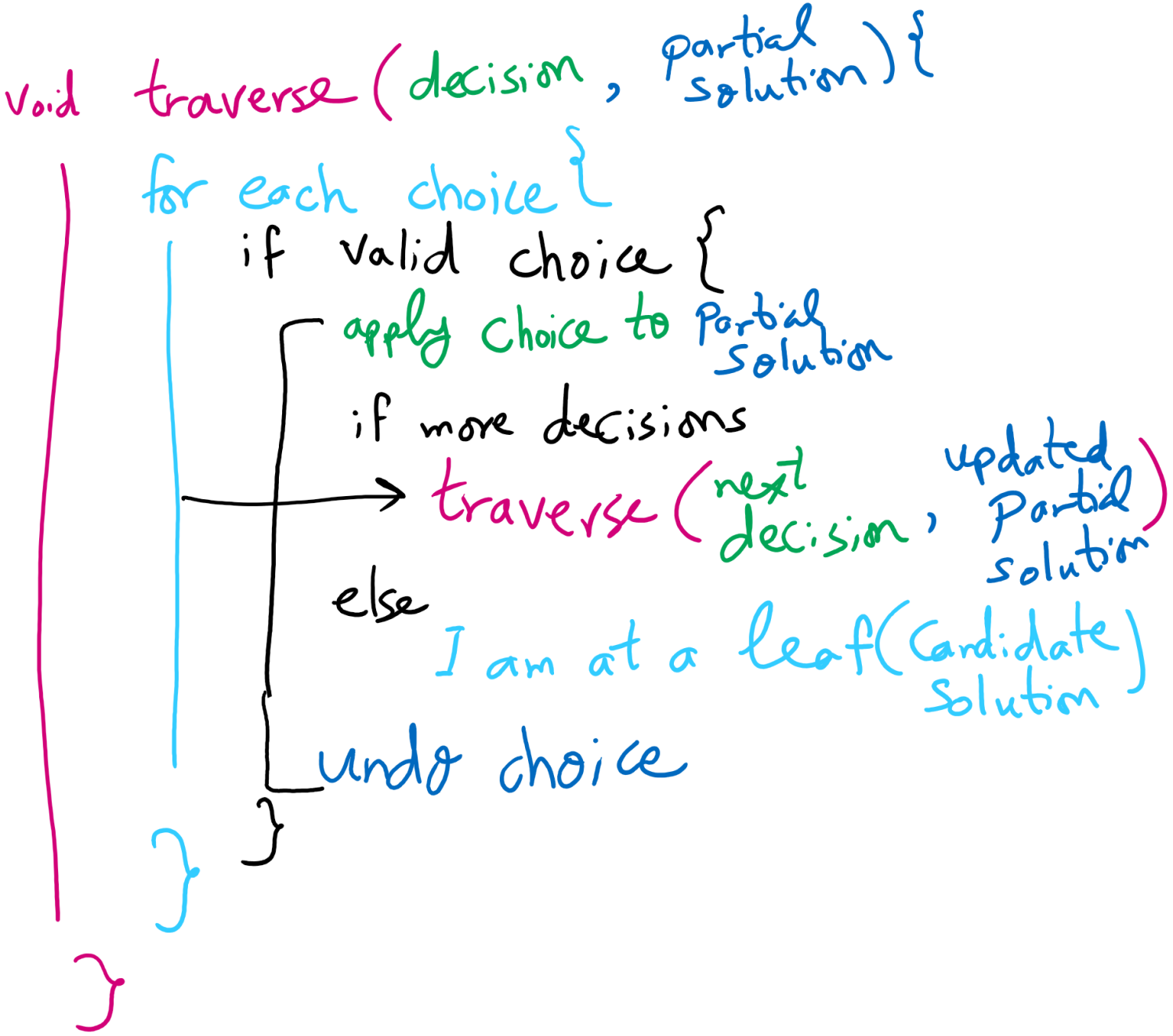
Are all 8 choices valid?

- Can't go past the edge of the board
- Can't reuse the same cells in the board
- Each letter added must lead to the prefix of an actual word
  - Check the dictionary as we add each letter, if there is no word with the currently constructed string as a prefix, don't go further down this way
  - Practically, this can be used for huge savings
- This is called pruning!

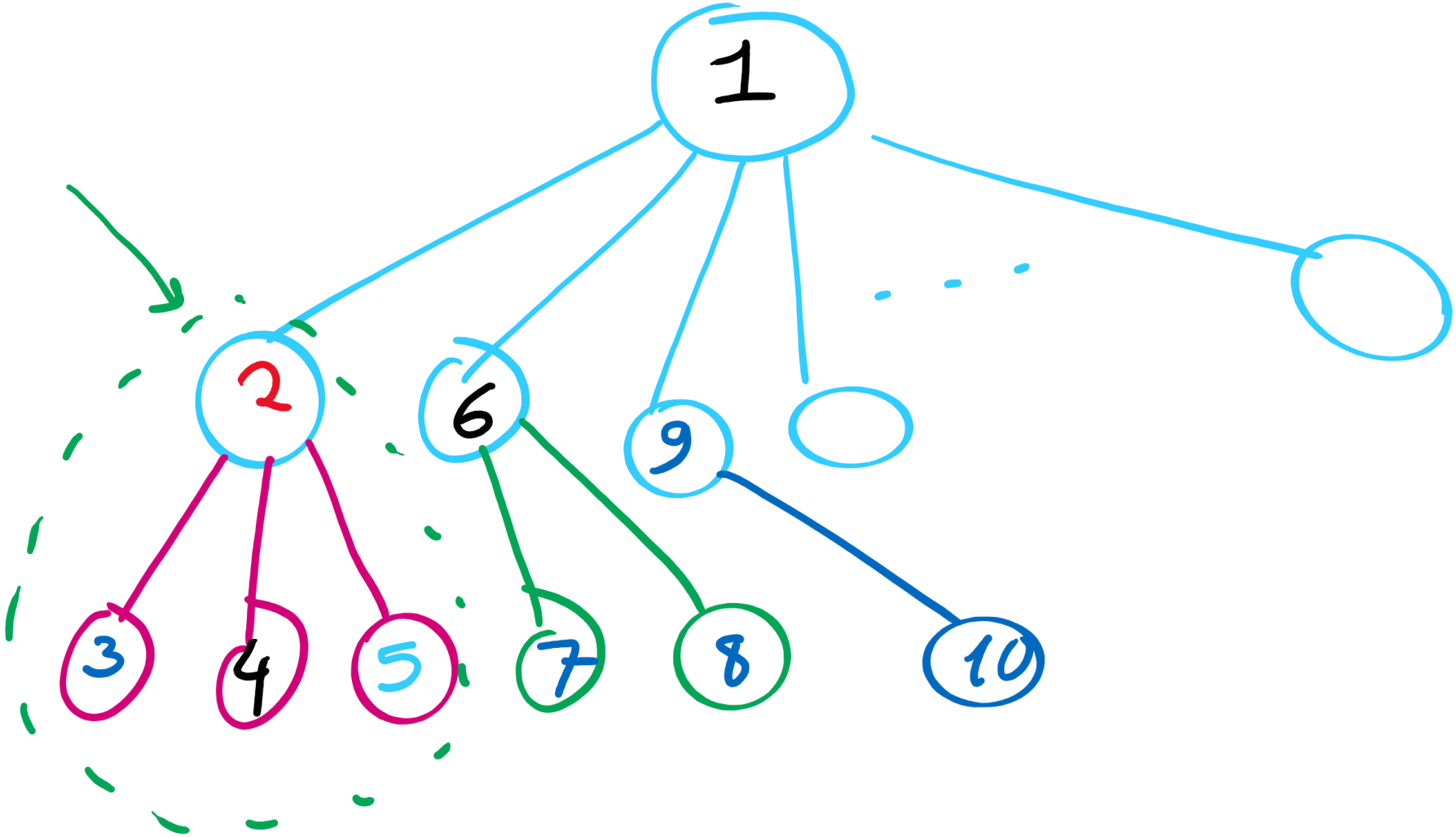
Then what?

- How can we code up such exhaustive search with pruning?
- The algorithm is called backtracking.

Backtracking Algorithm



Search Tree Traversal Order in Backtracking



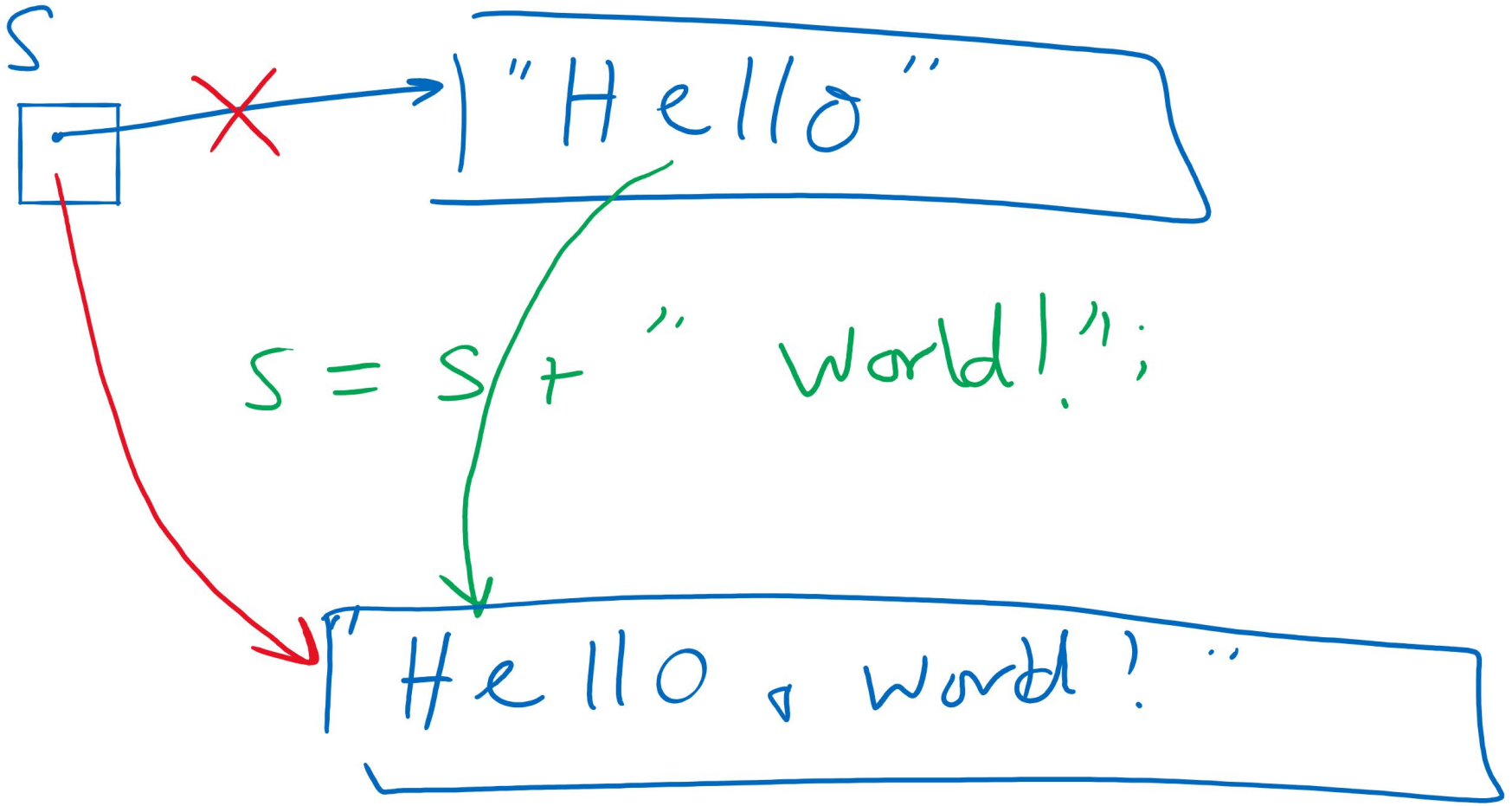
Search Tree Size

$$\#\text{leaves} = \Theta(\#\text{nodes}) = \Theta\left((\#\text{ branches per node})^{\#\text{ levels} - 1}\right)$$

Implementation concerns with Boggle

- Constructing the words over the course of recursion will mean building up and tearing down strings
  - Moving forward adds a new character to the current word string
  - Backtracking removes the most recent character
  - Basically pushing/popping to/from a string stack
- Push/Pop stack operations are generally $\Theta(1)$
  - Unless you need to resize, but that cost can be amortized
- Java Strings, however, are *immutable*
  - s = new String("Here is a basic string");
  - s = s + " this operation allocates and initializes all over again";
  - Becomes essentially a $\Theta(n)$ operation
    - Where n is the length of the string

Concatenating to A String Object



StringBuilder to the rescue

- append() and deleteCharAt() can be used to push and pop
  - Back to Θ(1)!
    - Still need to account for resizing, though…
- StringBuffer can also be used for this purpose
  - Differences?

Please submit your reflections by using the CourseMIRROR App

If you are having a problem with CourseMIRROR, please send an email to coursemirror.development@gmail.com